Computational calculus: bridging reduction and evaluation

(Article begins on next page)

Computational calculus: bridging reduction and evaluation

Claudia Faggian[1], Giulio Guerrieri[2], and Riccardo Treglia[3]

[1] Université de Paris, IRIF, CNRS, F-75013 Paris, France
faggian@irif.fr
[2] University of Bath, Department of Computer Science, Bath, UK
giulio.guerrieri@gmail.com
[3] Università di Torino, Department of Computer Science, Turin, Italy
riccardo.treglia@unito.it

Abstract

In Moggi's computational calculus, reduction is the contextual closure of the rules obtained by orienting three monadic laws. In the literature, evaluation is usually defined as the closure under weak contexts (no reduction under binders): $\mathsf{E} = \langle\rangle \mid \mathsf{let}\, x := \mathsf{E} \,\mathsf{in}\, M$.

We show that, when considering all the monadic rules, weak reduction is non-deterministic, non-confluent, and normal forms are not unique. However, when interested in returning a value (convergence), the only necessary monadic rule is β, whose evaluation is deterministic. The proof relies on tools coming from a calculus inspired by linear logic.

The *computational λ-calculus*, noted λ_c, was introduced by Moggi [12, 13, 14] as a meta-language to describe computational effects in programming languages. Since then, computational λ-calculi have been developed as foundations of programming languages, formalizing both functional and effectful features [21, 1, 15, 10, 2], in a still active line of research.

To model effectful features at a semantic level, Moggi used the categorical notion of *monad*. A monad can be equivalently presented as a Kleisli triple satisfying three identities [14, 11]. At an operational level, Moggi [12] internalized these identities into the syntax of λ_c, giving rise to three conversion rules—called *monadic laws*—that are added to the usual β and η rules.

Nowadays the literature is rich of computational calculi that refine Moggi's λ_c. Such calculi are presented in at least three different fashions: fully equational systems [10, 17] (all conversion rules are unoriented identities); hybrid systems where β (and η, if considered) are oriented rules while the monadic laws are identities on terms [2]; reduction systems where every rule is oriented [18]. Here we follow the latter approach, which brings to the fore operational aspects of reduction and evaluation which seem to have been neglected in the literature.

In the literature of calculi with effects [10, 2], *evaluation* is usually *weak*, that is, it is not allowed in the scope of the binders (λ or let). This is the way evaluation is implemented by functional programming languages such as Haskell and OCaml. Moreover, only β $\mathsf{let}.\beta$ are considered.

However, in Moggi's λ_c and in [18], the reduction is full, that is, reduction is the compatible closure of all the monadic rules. When considering all the rules, we observe that evaluation (*i.e.* weak reduction) is non-deterministic, non-confluent, and normal forms are not unique.

Reduction and Evaluation. Let us recall reduction in Sabry and Wadler's λ_{ml^*} [19, Sect. 5]—which we display in Figure 1—a refined variant of Moggi's untyped λ_c [12]. It has a two sorted syntax that separates *values* (*i.e.* variables and abstractions) and *computations*. The latter are either *let*-expressions (aka explicit substitutions, capturing monadic binding), or applications (of values to values), or coercions $[V]$ of values V into computations.

- The *reduction rules* in λ_{ml^*} are the usual β from Plotkin's *call-by-value* λ-calculus [16], plus the oriented version of three *monadic laws*: $\mathsf{let}.\beta$, $\mathsf{let}.\eta$, $\mathsf{let}.ass$ (see Figure 1).

$$\begin{array}{rrcl} \textit{Values:} & V, W & ::= & x \mid \lambda x.M \\ \textit{Computations:} & M, N & ::= & [V] \mid \mathsf{let}\, x \mathrel{:=} M \,\mathsf{in}\, N \mid VW \end{array}$$

Reduction rules:

$$\begin{array}{rrcll} (\beta) & (\lambda x.M)V & \mapsto_\beta & M[V/x] & \\ (\eta) & \lambda x.Vx & \mapsto_\eta & V & x \notin \mathsf{fv}(V) \\ (\mathsf{let}.\beta) & \mathsf{let}\, x \mathrel{:=} [V] \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.\beta} & N[V/x] & \\ (\mathsf{let}.\eta) & \mathsf{let}\, x \mathrel{:=} M \,\mathsf{in}\, [x] & \mapsto_{\mathsf{let}.\eta} & M & \\ (\mathsf{let}.ass) & \mathsf{let}\, y \mathrel{:=} (\mathsf{let}\, x \mathrel{:=} L \,\mathsf{in}\, M) \,\mathsf{in}\, N & \mapsto_{\mathsf{let}.ass} & \mathsf{let}\, x \mathrel{:=} L \,\mathsf{in}\, (\mathsf{let}\, y \mathrel{:=} M \,\mathsf{in}\, N) & x \notin \mathsf{fv}(N) \end{array}$$

Figure 1: λ_{ml^*}: Syntax and Reduction

- *Reduction* $\rightarrow$ is the compatible closure of the rules.

Following standard practice, we define *evaluation* $\underset{\mathsf{w}}{\rightarrow}$ (aka *sequencing*) as the closure of the rules under *evaluation context* E:

$$\mathsf{E} \ ::= \ \langle\,\rangle \ \mid \ \mathsf{let}\, x \mathrel{:=} \mathsf{E} \,\mathsf{in}\, M \qquad\qquad \text{evaluation context}$$

Despite the prominent role that weak reduction has in the literature of calculi with effects, what one obtain is somehow unexpected. While full reduction $\rightarrow_{ml^*}$ is confluent, the closure of the rules under *evaluation context* turns out to be *non-deterministic*, *non-confluent*, and its *normal forms* are *not unique*. Example 2 and Example 3—given at the end of the paper—demonstrate such points.

Note that the issues only come from the monadic rules $\mathsf{let}.\eta$ and $\mathsf{let}.ass$ (sometimes called *identity* and *associativity*, respectively, in the literature), not from β or $\mathsf{let}.\beta$. Note also that the literature on computational λ-calculi that studies weak reduction [10, 2] usually deals with the rules $\mathsf{let}.ass$ and $\mathsf{let}.\eta$ as unoriented identities, the only oriented rules being β and $\mathsf{let}.\beta$.

A bridge between Evaluation and Reduction. On the one hand, computational λ-calculi such as λ_{ml^*} have a unrestricted *non-deterministic reduction* that generates the equational theory of the calculus, studied for foundational and semantic purposes. On the other hand, *weak reduction* has a prominent role in the literature of computational λ-calculi, because it models an ideal programming language. Indeed, when restricted to closed terms (which are the terms corresponding to programs), normal forms of weak reduction coincide with values; and when restricted to β and $\mathsf{let}.\beta$ steps, weak reduction is deterministic and corresponds to abstract machines implementing programming languages. It is then natural to wonder what is the relation between reduction and evaluation.

In Plotkin's call-by-value λ-calculus [16], the following *convergence result* provides a bridge between reduction and evaluation: if a term M β-reduces to a value, then M only needs *weak* β-reduction to reach a value.

$$M \rightarrow_\beta^* V \text{ (for some value } V) \iff M \underset{\mathsf{w}}{\rightarrow}_\beta^* V' \text{ (for some value } V') \tag{1}$$

In λ_{ml^*}, despite several drawbacks of weak reduction, we can still prove a *convergence* result similar to (1) relating reduction and evaluation: to reach a value in λ_{ml^*}, weak β-steps and weak $\mathsf{let}.\beta$-steps suffice.

Theorem 1 (Convergence)**.** *Let M be a computation in λ_{ml^*} and let $\to_{ml^*_-} := \to_{ml^*} \smallsetminus \to_\eta$.*

$$M \to^*_{ml^*_-} [V] \text{ (for some value } V\text{)} \iff M \underset{\mathsf{w}}{\rightarrow}^*_{\beta,\, \mathsf{let}.\beta} [V'] \text{ (for some value } V'\text{)} \tag{2}$$

The proof is non-trivial. We rely on tools inspired by linear logic. More precisely, in [7] we study the reduction theory of a closely related calculus, namely $\lambda_{©}$ in [3], and then transfer the convergence to λ_{ml^*}, via translation.

Conclusion. Convergence in λ_{ml^*} relates full reduction to evaluation, and provides a theoretical justification to the following facts:

1. that functional programming languages with computational effects use weak reduction as evaluation mechanism; indeed, weak reduction is enough to *return values*.
2. that in computational λ-calculi, when interested in returning a value, the only *rules* of interest *for weak reduction* are β and $\mathsf{let}.\beta$—which are deterministic and do not have unpleasant rewriting properties—while the rules $\mathsf{let}.ass$ and $\mathsf{let}.\eta$ can be safely considered as unoriented identities external to the reduction.

Examples

(Non-)Confluence.

Example 2 (Non-confluence)**.** Let M be a computation in normal form, for instance $M = xx$.

$$\begin{tikzcd}
\mathsf{let}\, y := (\mathsf{let}\, x := zz \,\mathsf{in}\, M) \,\mathsf{in}\, [y] \arrow[r, "\mathsf{let}.\eta", "\mathsf{w}"'] \arrow[d, "\mathsf{let}.ass"', "\mathsf{w}"] & \mathsf{let}\, x := zz \,\mathsf{in}\, M \\
\mathsf{let}\, x := zz \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, [y]) &
\end{tikzcd}$$

Both $\mathsf{let}\, x := zz \,\mathsf{in}\, M$ and $\mathsf{let}\, x := zz \,\mathsf{in}\, (\mathsf{let}\, y := M \,\mathsf{in}\, [y])$ are normal for $\underset{\mathsf{w}}{\rightarrow}_{ml^*}$ (in the latter, the $\mathsf{let}.\eta$-redex $\mathsf{let}\, y := M \,\mathsf{in}\, [y]$ cannot be fired by weak reduction), but they are distinct.

Example 3 (Non-confluence)**.** Non-confluence and non-uniqueness of normal forms of $\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass}$, of $\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass} \cup \underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.\beta} \cup \underset{\mathsf{w}}{\rightarrow}_{\beta}$, and of $\underset{\mathsf{w}}{\rightarrow}_{ml^*}$. Let $R = P = Q = L = zz$ and:

$$M \;:=\; \overline{\mathtt{let}\ z = \underline{(\mathtt{let}\ x = (\mathtt{let}\ y = L\ \mathtt{in}\ Q)\ \mathtt{in}\ P)}\ \mathtt{in}\ R}$$

There are two weak $\mathsf{let}.ass$-redexes, the overlined one and the underlined one. So,

$$\begin{aligned}
M &\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass} \mathsf{let}\, x := (\mathsf{let}\, y := L \,\mathsf{in}\, Q) \,\mathsf{in}\, (\mathsf{let}\, z := P \,\mathsf{in}\, R) \\
&\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass} \mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, x := Q \,\mathsf{in}\, (\mathsf{let}\, z := P \,\mathsf{in}\, R)) =: M'
\end{aligned}$$

$$\begin{aligned}
M &\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass} \mathsf{let}\, z := (\mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, x := Q \,\mathsf{in}\, P)) \,\mathsf{in}\, R \\
&\underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.ass} \mathsf{let}\, y := L \,\mathsf{in}\, (\mathsf{let}\, z := (\mathsf{let}\, x := Q \,\mathsf{in}\, P) \,\mathsf{in}\, R) =: M''
\end{aligned}$$

Both M' are M'' are normal for $\underset{\mathsf{w}}{\rightarrow}_{ml^*}$ (in M'', the $\mathsf{let}.ass$-redex $\mathsf{let}\, z := (\mathsf{let}\, x := Q \,\mathsf{in}\, P) \,\mathsf{in}\, R$ is under the scope of a let and so cannot be fired by weak reduction), but they are distinct.

(Non-)Factorization. Another aspect making the theory of reduction for λ_{ml^*} (and for other computational λ-calculi) tricky to study is the lack of *factorization*, which is the simplest possible form of *standardization*.

In Plotkin's call-by-value λ-calculus [16] (which can be seen as the restriction of λ_{ml^*} where the reduction is generated only by the β-rule), weak reduction satisfies factorization, that is any reduction sequence can be *reorganized* as weak steps followed by non-weak steps:

$$\rightarrow_\beta^* \subseteq \underset{\mathsf{w}}{\rightarrow}_\beta^* \cdot \underset{\neg\mathsf{w}}{\rightarrow}_\beta^* \tag{3}$$

But in λ_{ml^*} (and other computational λ-calculi), weak factorization *does not hold* as shown by the following counterexample[1].

Example 4 (Non-factorization)**.** Consider

$$M \coloneqq \mathsf{let}\, y \coloneqq (zz)\, \mathsf{in}\, (\mathsf{let}\, x \coloneqq [y]\, \mathsf{in}\, [x]) \underset{\neg\mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} \mathsf{let}\, y \coloneqq (zz)\, \mathsf{in}\, [y] \underset{\mathsf{w}}{\rightarrow}_{\mathsf{let}.\eta} (zz) \eqqcolon N$$

Weak steps are not possible from M, so it is *impossible* to factorize the reduction form M to N as $M \underset{\mathsf{w}}{\rightarrow}_{ml^*}^* \cdot \underset{\neg\mathsf{w}}{\rightarrow}_{ml^*}^* N$.

References

[1] Nick Benton, John Hughes, and Eugenio Moggi. Monads and effects. In *Applied Semantics, International Summer School, APPSEM 2000*, volume 2395 of *Lecture Notes in Computer Science*, pages 42–122. Springer, 2002.

[2] Ugo Dal Lago, Francesco Gavazzo, and Paul B. Levy. Effectful Applicative Bisimilarity: Monads, Relators, and Howe's Method. In *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017*, pages 1–12. IEEE Computer Society, 2017.

[3] Ugo de'Liguoro and Riccardo Treglia. The untyped computational λ-calculus and its intersection type discipline. *Theor. Comput. Sci.*, 846:141–159, 2020.

[4] Thomas Ehrhard. Call-by-push-value from a linear logic point of view. In *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016 Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 202–228. Springer, 2016.

[5] Thomas Ehrhard and Giulio Guerrieri. The bang calculus: an untyped lambda-calculus generalizing call-by-name and call-by-value. In *Proceedings of the 18th International Symposium on Principles and Practice of Declarative Programming (PPDP 2016)*, pages 174–187. ACM, 2016.

[6] Claudia Faggian and Giulio Guerrieri. Factorization in call-by-name and call-by-value calculi via linear logic. In *Foundations of Software Science and Computation Structures - 24th International Conference, FOSSACS 2021*, volume 12650 of *Lecture Notes in Computer Science*, pages 205–225. Springer, 2021.

[7] Claudia Faggian, Giulio Guerrieri, Ugo de'Liguoro, and Riccardo Treglia. On reduction and normalization in the computational core. *CoRR*, abs/2104.10267, 2021. Submitted to Math. Struct. Comp. Sci., special issue of IWC 2020.

[8] Giulio Guerrieri and Giulio Manzonetto. The bang calculus and the two Girard's translations. In *Proceedings Joint International Workshop on Linearity & Trends in Linear Logic and Applications (Linearity-TLLA 2018)*, volume 292 of *EPTCS*, pages 15–30, 2019.

[9] Paul Blain Levy. Call-by-push-value: A subsuming paradigm. In *Typed Lambda Calculi and Applications, 4th International Conference (TLCA'99)*, volume 1581 of *Lecture Notes in Computer Science*, pages 228–242, 1999.

[1] This counterexample was noticed by van Oostrom and sent by private communication.

[10] Paul Blain Levy, John Power, and Hayo Thielecke. Modelling environments in call-by-value programming languages. *Information and Computation*, 185(2):182 – 210, 2003.

[11] Saunders MacLane. *Categories for the Working Mathematician.* Graduate Texts in Mathematics. Springer, 2 edition, 1997.

[12] Eugenio Moggi. Computational Lambda-calculus and Monads. Report ECS-LFCS-88-66, University of Edinburgh, Edinburgh, Scotland, October 1988.

[13] Eugenio Moggi. Computational lambda-calculus and monads. In *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS '89)*, pages 14–23. IEEE Computer Society, 1989.

[14] Eugenio Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

[15] G. D. Plotkin and J. Power. Algebraic operations and generic effects. *Applied Categorical Structures*, 11(1):69–94, 2003.

[16] Gordon D. Plotkin. Call-by-name, call-by-value and the lambda-calculus. *Theor. Comput. Sci.*, 1(2):125–159, 1975.

[17] Gordon D. Plotkin and John Power. Notions of computation determine monads. In *Foundations of Software Science and Computation Structures, 5th International Conference, FOSSACS 2002*, volume 2303 of *Lecture Notes in Computer Science*, pages 342–356. Springer, 2002.

[18] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[19] Amr Sabry and Philip Wadler. A reflection on call-by-value. *ACM Trans. Program. Lang. Syst.*, 19(6):916–941, 1997.

[20] Alex Simpson. Reduction in a linear lambda-calculus with applications to operational semantics. In Jürgen Giesl, editor, *Term Rewriting and Applications*, pages 219–234, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[21] Philip Wadler and Peter Thiemann. The marriage of effects and monads. *ACM Trans. Comput. Log.*, 4(1):1–32, 2003.